Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Ingeniería de Software

Grupo 04

Ing. Orlando Zaldívar Zamorategui

TUTORIAL IS 2025-2 1. Introducción a la Ingeniería de Software. Definición de Ingeniería de Software.

EQUIPO 2

Benítez Guerrero Alexis
Cabrera Marcos Sergio Ivan
Moreno Muñiz Luis Gerardo

Ciudad Universitaria, Ciudad de México, a 27 / 05 / 2025

ÍNDICE

PÁG.

TUTORIAL IS 2025-1 1. Introducción a la Ingeniería de Software.
Definición de Ingeniería de Software.

1. OBJETIVO

El objetivo de este trabajo es proporcionar material informativo a la comunidad estudiantil de la Facultad de Ingeniería sobre el concepto de software y la ingeniería de software. Al finalizar, los estudiantes comprenderán la naturaleza intangible del software, los principios fundamentales que rigen su desarrollo y los desafíos únicos que enfrenta la ingeniería de software. Esto se logrará a través de un análisis detallado de los procesos, metodologías y prácticas específicas de esta disciplina, permitiendo a los participantes aplicar este conocimiento en situaciones reales y en su formación profesional futura.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 1. Personas construyendo una idea.

2. INTRODUCCIÓN

En la actualidad, la ingeniería de software se ha consolidado como una disciplina fundamental en el desarrollo de tecnologías que transforman nuestra vida diaria. A diferencia de otras ramas de la ingeniería, que se enfocan en la creación de productos físicos, la ingeniería de software se centra en la producción de software: un producto intangible que requiere un enfoque distintivo en su desarrollo y mantenimiento. Esta disciplina no solo aborda la creación de soluciones funcionales, sino que también exige una adaptación constante a un entorno tecnológico en rápida evolución. A lo largo de este trabajo, se explorarán los principios, metodologías y desafíos que caracterizan a la ingeniería de software, así como su interrelación con otras disciplinas y su impacto en el ámbito profesional.

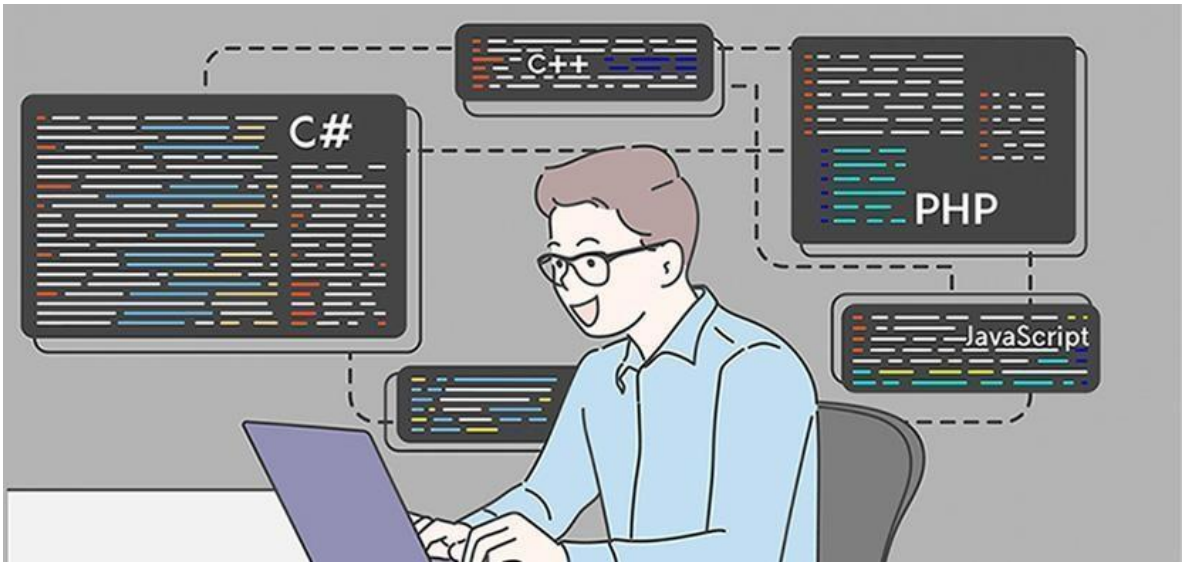

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 2. Hombre escribiendo código en una computadora.

3. DESARROLLO

3.1. ASPECTOS GENERALES DEL SOFTWARE

3.1.1. DEFINICIÓN

Se denomina software a todo aquello que es intangible en una computadora, abarcando tanto el conjunto de programas que especifican las instrucciones que el sistema debe seguir para funcionar (también conocido como código) como los datos que la máquina procesa y guarda.

Más formalmente, el IEEE define software como: *"El conjunto de los programas de computación, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo"*.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 3. Representación gráfica de software.

3.1.2. ¿DE QUÉ SE COMPONE?

El software se compone de:

1) Instrucciones o programas que, al ejecutarse, proporcionan las características, funciones y rendimiento esperados.
2) Estructuras de datos que permiten manipular información de manera efectiva.
3) Información explicativa, tanto en formatos impresos como digitales, que describe el funcionamiento y uso de los programas.

Sin embargo, el software abarca no solo los programas en sí, sino también la documentación asociada y la configuración de datos necesarios para que estos programas operen adecuadamente. Normalmente, un sistema de software incluye varios programas independientes, archivos de configuración que permiten su ejecución, un sistema de documentación que detalla la estructura del sistema, guías para el usuario que explican cómo utilizarlo y sitios web donde los usuarios pueden descargar información sobre actualizaciones o productos recientes.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 4. Personas diseñando software.

3.1.3. ¿CÓMO SE CLASIFICA EL SOFTWARE?

La clasificación del software es esencial para situar el ámbito de aplicación en el que se desarrollará una solución tecnológica. Según la clasificación propuesta por Roger Pressman, se pueden identificar diversas categorías principales en función del propósito y el entorno de aplicación del software.

1. **Software de sistemas.**

Este software es la base que permite la ejecución de otros programas. Incluye sistemas operativos, compiladores y programas que gestionan recursos y hardware, haciendo que otros desarrolladores puedan construir aplicaciones sobre ellos. Su característica principal es la estrecha interacción con el hardware, enfocándose en optimizar el acceso y la eficiencia de otros programas o usuarios.

2. **Software de aplicación.**

Son soluciones específicas que atienden las necesidades de organizaciones o individuos. Este tipo de software puede diseñarse para satisfacer los requisitos exclusivos de un cliente o desarrollarse como un producto de propósito general, utilizado por múltiples clientes que seleccionan el conjunto de funciones que mejor se adapta a sus necesidades. Ejemplos incluyen sistemas de gestión empresarial y ERP (Enterprise Resource Planning) para optimizar procesos de negocio.

3. **Software científico y de ingeniería.**

Estos programas se especializan en cálculos complejos y simulaciones avanzadas, utilizando modelos y algoritmos matemáticos detallados. Se emplean en áreas como la investigación científica, la ingeniería, y simulaciones de sistemas complejos como reacciones químicas o predicciones meteorológicas.

4. **Software incrustado o empotrado.**

Se encuentra integrado en dispositivos físicos y permite controlar su funcionamiento. Desde electrodomésticos hasta sistemas automotrices, este tipo de software está diseñado para cumplir tareas específicas en tiempo real, a menudo con limitaciones de recursos y adaptado al hardware particular que opera.

5. **Software de línea de productos.**

Orientado al consumidor, este software proporciona una funcionalidad específica y generalizada para una amplia variedad de usuarios. Dentro de esta categoría se incluyen aplicaciones como hojas de cálculo, procesadores de texto y software de contabilidad, que ofrecen capacidades comunes para satisfacer necesidades compartidas.

6. **Aplicaciones web.**

Conocidas como "webapps", estas aplicaciones han evolucionado para ser mucho más que simples archivos de hipertexto. Integran bases de datos y servicios complejos en entornos distribuidos, permitiendo realizar operaciones bancarias, gestionar compras, o jugar en línea. La conveniencia, rapidez y accesibilidad de las aplicaciones web son elementos clave de su éxito.

7. **Software de inteligencia artificial.**

Este software incorpora técnicas avanzadas y algoritmos para resolver problemas complejos, frecuentemente de naturaleza no numérica. Incluye áreas como sistemas expertos, visión artificial, redes neuronales, y reconocimiento de patrones, permitiendo que las máquinas aprendan y actúen en escenarios diversos.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 5. Representación gráfica de la diversidad en el software.

3.1.4. VENTAJAS, DESVENTAJAS Y DESAFÍOS

El software ha transformado radicalmente nuestra vida cotidiana y la economía global, influyendo en cómo interactuamos, trabajamos y consumimos información.

Desde el impulso a nuevos modelos de negocio y la creación de tecnologías revolucionarias hasta el desarrollo de aplicaciones que facilitan la comunicación y la gestión de datos, el software está presente en casi todas las áreas de la vida moderna.

A pesar de sus enormes beneficios, el desarrollo de software enfrenta también múltiples desafíos.

Estos incluyen la gestión de costos y tiempos, la necesidad de pruebas exhaustivas para reducir fallos, y el esfuerzo constante de mantenimiento, lo que hace que el proceso sea complejo y demandante.

Las fallas en el software pueden tener repercusiones significativas en sectores críticos, subrayando la responsabilidad de los desarrolladores para crear productos seguros, confiables y éticos.

A continuación, mostraremos ventajas, desventajas y desafíos de lo dicho anteriormente.

Ventajas

1) **Transformación de modelos de negocio.**

El software ha sido clave en la creación de nuevos modelos empresariales innovadores, como las plataformas de transporte y las ventas en línea, que han cambiado la manera en que las personas interactúan y adquieren bienes y servicios.

2) **Impulso de tecnologías innovadoras.**

Ha facilitado el desarrollo de avances en áreas como la ingeniería genética, permitiendo nuevas posibilidades en medicina y biotecnología.

3) **Expansión de tecnologías existentes.**

El software ha evolucionado las tecnologías de comunicación, como los mensajes en redes celulares, habilitando la creación de servicios como mensajería instantánea, redes sociales, y herramientas de atención al cliente y marketing.

4) **Digitalización de contenidos.**

Ha contribuido a la obsolescencia de tecnologías tradicionales (como impresiones y CDs) mediante opciones digitales más sostenibles. Gracias al software, el acceso a libros, música y películas en formato digital se ha vuelto más rápido y accesible.

5) **Avances en inteligencia artificial y análisis de datos.**

El software potencia la inteligencia artificial y facilita el procesamiento de grandes volúmenes de datos, permitiendo conocimientos más detallados y eficaces, incluso reemplazando algunas prácticas laborales tradicionales como las auditorías por muestreo.

6) **Disponibilidad y accesibilidad.**

El software ha pasado de ser un producto empresarial a estar ampliamente disponible como producto comercial, tanto en tiendas físicas como en plataformas digitales para descarga directa.

7) **Importancia económica.**

Las empresas de desarrollo de software dominan sectores económicos, siendo la mayoría de las empresas más grandes del mundo negocios de software o hardware que dependen de él.

8) **Doble valor.**

El software no solo es un producto en sí mismo, sino que también es esencial para gestionar la información, uno de los recursos más valiosos hoy en día.

9) Acceso a la información y conectividad global.

Gracias al software que potencia Internet, es posible acceder instantáneamente a información y comunicarse de manera global, lo que transforma permanentemente nuestras formas de interacción social y comercial.

Desventajas

1) **Impacto de fallas y errores.**

Los errores en el software pueden tener consecuencias graves en sectores clave como el empresarial, bancario, gubernamental, y de servicios críticos, afectando tanto a la economía como a la seguridad de personas e instituciones.

2) **Responsabilidad de los desarrolladores.**

Los profesionales de software tienen una responsabilidad significativa en garantizar que los productos sean seguros y confiables, pues las fallas pueden comprometer directamente la salud, seguridad, y derechos de los usuarios.

Desafíos

1) **Tiempo de desarrollo prolongado.**

Crear software puede requerir más tiempo del estimado, ya que no es un proceso predecible y puede enfrentar demoras debido a su complejidad.

2) **Dificultades en la estimación de costos y plazos.**

Es complicado prever con precisión el presupuesto y el tiempo que se necesitará, ya que a menudo surgen cambios y requerimientos adicionales.

3) **Medición de progreso.**

Evaluar el avance en proyectos de software es difícil debido a su naturaleza intangible y la interdependencia de componentes.

4) **Altos costos de desarrollo.**

Los proyectos de software suelen exceder los presupuestos iniciales, con necesidades de presupuesto que no siempre se conocen de antemano.

5) **Garantía de calidad limitada.**

A pesar de las pruebas exhaustivas, siempre existe la posibilidad de que algunos defectos pasen desapercibidos.

6) **Elevados costos de mantenimiento.**

La mayoría del esfuerzo en el software se destina a su mantenimiento posterior a la entrega, representando entre el 60% y el 80% del trabajo total.

7) **Expectativas de tiempos de entrega cortos.**

Los clientes demandan tiempos de entrega cada vez menores, esperando soluciones rápidas, lo que añade presión sobre el desarrollo.

3.2. INGENIERÍA DE SOFTWARE

3.2.1. ¿QUÉ ES? ¿DE QUÉ SE TRATA?

La ingeniería de software es una disciplina dedicada al diseño, desarrollo y entrega de sistemas de software que cumplen con los requisitos del cliente de forma eficaz y rentable. Este campo no solo se centra en la creación de un programa, sino que abarca todas las etapas del proceso, desde la comunicación inicial con el cliente hasta la planificación y gestión del proyecto, e incluso el mantenimiento posterior a la entrega. Como disciplina, la ingeniería de software involucra aplicar teorías, métodos y herramientas que ayuden a resolver problemas específicos, aunque a veces no existan soluciones o métodos predefinidos para ellos. Además, los ingenieros de software son conscientes de las limitaciones financieras y organizativas, y buscan soluciones que respeten estas restricciones. Así, la ingeniería de software no solo aborda los aspectos técnicos del desarrollo, sino también la gestión de proyectos, la creación de métodos y herramientas, y el mantenimiento continuo del producto a lo largo de su vida útil.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 6. Equipo llevando a cabo un proceso de ingeniería de software.

En conjunto, la ingeniería de software abarca todos los aspectos de la producción de software, desde la especificación inicial hasta el mantenimiento, asegurando que el sistema entregue valor de manera continua y ajustada a las necesidades del cliente. A continuación, mostraremos cómo algunos autores, libros e instituciones definen a la ingeniería de software.

"Es el establecimiento y uso de principios sólidos de la ingeniería para obtener económicamente un software confiable y que funcione de modo eficiente en máquinas reales". [Bauer, 1972]

"Es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software". [Zelkovitz, 1978]

"Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo operación (funcionamiento) y mantenimiento del software: es decir, la aplicación de ingeniería al software". [IEEE, 1993]

"Es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema hasta el mantenimiento de este después que se utiliza". [Sommerville, 2004]

"Es una disciplina que integra el proceso, los métodos, y las herramientas para el desarrollo de software de computadora". [Pressman, 2005]

"Ingeniería de software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software". [Bohem, 1976]

"Un proyecto de ingeniería de software es una actividad que requiere recursos para crear un producto de software. Este producto final no solo incluye el programa mismo, sino también toda la documentación, distribución y empaquetado relacionados. Tal proyecto se sustenta en los mismos recursos que cualquier otro proyecto de desarrollo de productos: tiempo, dinero y personal. Dependiendo del tamaño del proyecto, la cantidad de estos recursos puede ser considerable y, en ocasiones, compleja de gestionar. Una mala administración de estos recursos puede resultar muy costosa e incluso llevar a la cancelación del proyecto. Para lograr un desarrollo eficiente y exitoso, el proyecto de ingeniería de software aplica los principios de la ingeniería de software, dividiéndose en fases que optimizan el avance ordenado del proyecto; en algunos casos, también se subdividen en proyectos más pequeños. Esta fragmentación hace que un proyecto grande sea más manejable y facilita el uso eficaz de los recursos. Aunque todo proyecto de ingeniería de software corre el riesgo de errores y mala administración, una buena organización y gestión ayudan a minimizar estos problemas, reduciendo el riesgo de fracaso o cancelación". (Sommerville, 2004)

La ingeniería de software se estructura en varias capas esenciales. Como se observa en la figura 1, un enfoque de ingeniería —incluyendo el de software— debe sustentarse en un compromiso organizacional hacia la calidad. Filosofías como la administración total de la calidad y Six Sigma promueven una cultura de mejora continua, que impulsa la evolución de métodos más eficientes en ingeniería de software. Este compromiso con la calidad es la base sobre la que se asienta la ingeniería de software.

En el centro de esta estructura se encuentra la capa de proceso, la cual actúa como el eje integrador de las capas tecnológicas, facilitando el desarrollo de software de manera lógica y oportuna. El proceso establece una estructura que permite emplear tecnologías de ingeniería de software eficazmente. A su vez, este proceso de software proporciona la base para gestionar los proyectos, enmarcando el contexto en el cual se aplican métodos técnicos, se producen productos de trabajo (como modelos, documentos y reportes), se fijan puntos de referencia, se asegura la calidad y se gestiona el cambio adecuadamente.

Los métodos de ingeniería de software aportan el conocimiento técnico necesario para desarrollar software y abarcan una variedad de actividades, como la comunicación, el análisis de requisitos, la modelación de diseños, la construcción de programas, las pruebas y el soporte. Estos métodos se fundamentan en principios básicos que guían cada área tecnológica y engloban tanto actividades de modelado como otras técnicas descriptivas.

Por último, las herramientas de ingeniería de software ofrecen soporte automatizado o semiautomatizado para los procesos y métodos. Cuando estas herramientas se integran de manera que la información generada por una pueda ser usada por otra, se crea un entorno llamado ingeniería de software asistido por computadora, que facilita el desarrollo de software.

Figura 1: Esquema de la jerarquía de herramientas de ingeniería de software.

"La ingeniería de software abarca mucho más que la programación, y del mismo modo, los participantes en un proyecto de este tipo no se limitan a programadores. La amplia variedad de tareas que puede requerir el proyecto demanda un conjunto diverso de habilidades provenientes de diferentes áreas". [Bruegge y Dutoit, 2004]

El concepto de proyecto de ingeniería de software clasifica a todos los participantes como "partes interesadas", desde los directivos hasta los propios clientes. No hay restricciones en cuanto al tipo de proyectos de software que un cliente puede solicitar, y esta diversidad significa que se pueden necesitar habilidades de prácticamente cualquier profesión como parte del proceso de desarrollo de software.

3.2.2. ¿CUÁL ES EL PROCESO A SEGUIR PARA DESARROLLAR SOFTWARE?

El proceso de desarrollo de software consiste en convertir las necesidades del usuario en requerimientos, estos en un diseño, y luego en código que se prueba, documenta y certifica para su uso operativo [Jacobson 1998].

En términos generales, el desarrollo de software sigue una serie de etapas interconectadas que comienzan con el primer contacto con el cliente y culminan con la implementación del sistema. En ocasiones, es necesario actualizar el software, repitiendo así el proceso en versiones posteriores. Estas etapas, que se ilustran en el siguiente esquema, son:

1. **Definición Inicial**: Se establece una visión general del proyecto, incluyendo los resultados esperados, un primer presupuesto y los acuerdos preliminares necesarios para iniciar el trabajo.

2. **Análisis de Requerimientos y Viabilidad**: Una vez iniciado el proyecto, se recopilan y evalúan detalladamente los requerimientos del cliente, incluyendo restricciones que puedan existir. Esta fase también permite determinar la viabilidad del proyecto.
3. **Diseño General y Detallado**: Basándose en los requerimientos, se elabora un diseño de la aplicación que satisfaga las necesidades del cliente. Este diseño empieza con una arquitectura general de la aplicación, seguida de un diseño detallado de sus componentes, similar a un plano arquitectónico que guiará la construcción del software.
4. **Codificación**: Los componentes diseñados se implementan mediante programación.
5. **Pruebas**: El software se verifica para asegurar que no tenga errores y que cumpla con los requisitos definidos originalmente por el usuario.
6. **Implementación:** El software se instala en los sistemas del cliente, y se asegura que los usuarios reciban la capacitación necesaria para operarlo adecuadamente.
7. **Evolución:** Después de la implementación, el software suele requerir ajustes, ya sea para corregir errores no detectados previamente o para añadir nuevas funcionalidades.

También podemos verlo como 5 casos, los cuales son:

- **Planificación:** es esencial en el desarrollo de software, ya que permite organizar tareas complejas y asignar roles. En lugar de ser solo una fase, la planificación se mantiene durante todo el proceso, ajustándose a cambios de requisitos y especificaciones. Este enfoque continuo facilita un proceso adaptable que permite definir claramente cada módulo del sistema y sus plazos, mejorando la estimación de costos y tiempos (Schach, 2008).
- **Resolución de problemas:** es el núcleo de la ingeniería de software. Cada software es una solución a un problema específico planteado por el cliente, como crear una aplicación de procesamiento de texto o una tienda en línea. Este proceso no es una simple conversión de problema a solución; requiere analizar el desafío, explorar alternativas, y diseñar una solución viable que luego se implemente en el software (Blum, 1992).
- **Modelado:** actúa como puente entre la idea de solución y el código. Crear un modelo ayuda a los ingenieros a abstraer y simplificar los aspectos complejos de un sistema, permitiendo planificar su estructura antes de codificar. Estos modelos incluyen el entorno del sistema, el problema inicial y

las soluciones planteadas. Los ingenieros utilizan técnicas como el Lenguaje Unificado de Modelado (UML), que permite visualizar el sistema y comprender cómo se integran sus partes (Booch et al., 1999; Bruegge y Dutoit, 2004).

- **Comunicación:** es constante en un proyecto de software. Los clientes deben comunicar sus necesidades, los líderes de equipo deben entenderlas y los desarrolladores deben aclarar cualquier duda con la gerencia. Esta comunicación, que se realiza a través de reuniones periódicas y revisiones, garantiza la comprensión del proyecto y permite monitorear el avance y gestionar adecuadamente las expectativas del cliente (Bruegge y Dutoit, 2004).
- **Gestión de recursos:** involucra la administración de tiempo, dinero, personal y equipos que se usan a lo largo del proyecto. Estos recursos se monitorean y ajustan según el progreso y las necesidades específicas de cada etapa, asegurando que el proyecto mantenga la eficiencia y se completen las tareas planificadas dentro de los límites establecidos.

Este proceso general se complementa con un conjunto de actividades adicionales, como la comunicación continua con el cliente, la planeación detallada, la iteración para mejoras incrementales y la incorporación de actividades sombrilla, que ayudan a gestionar y optimizar cada etapa.

Las actividades sombrilla, esenciales en el desarrollo de software, apoyan al equipo para controlar y gestionar el progreso, calidad, cambios y riesgos a lo largo de un proyecto. Estas actividades incluyen:

- **Seguimiento y control del proyecto:** permite que el equipo de software monitoree el progreso comparando los avances con el plan del proyecto y tomando acciones correctivas ante cualquier desviación detectada.
- **Gestión de riesgos:** implica una evaluación continua de los riesgos que puedan impactar el proyecto o la calidad del producto, así como la implementación de acciones para reducir su probabilidad o impacto.
- **Aseguramiento de la calidad del software:** establece normas, estándares y protocolos para asegurar que el software final cumpla con los niveles de calidad requeridos.
- **Revisiones técnicas:** consiste en evaluar productos de trabajo en cada etapa para identificar y corregir errores antes de que avancen a fases posteriores.

- **Mediciones y métricas:** se definen y recolectan métricas que ayudan al equipo a evaluar el estado del proyecto, identificar posibles correcciones y contribuir al proceso de mejora continua.
- **Gestión de la configuración del software:** permite administrar los efectos de cambios en el software a lo largo de todo el proceso de desarrollo, manteniendo un control sobre las modificaciones.
- **Gestión de la reutilización:** establece criterios para identificar y reutilizar productos de trabajo, incluidos componentes de software, desde el inicio, asegurando su disponibilidad para proyectos futuros.
- **Preparación y producción de productos de trabajo:** comprende las actividades necesarias para crear documentos, modelos, registros, plantillas y listas que se requieren en cada fase del proyecto.

Figura 2: Esquema del proceso de desarrollo de software.

Figura 3: Esquema del proceso de desarrollo de software haciendo énfasis en lo relacionado con la calidad.

3.2.3. ¿TIENE PRINCIPIOS?

A pesar de que la ingeniería de software aún carece de principios fundamentales universalmente reconocidos (Bourque et al., 2002), se han realizado varios esfuerzos para establecerlos (Boehm, 1983; Davis, 1995). Uno de los textos más citados en esta área, Fundamentals of Software Engineering de Ghezzi et al. (2002), con más de 1650 referencias, propone siete principios esenciales que deben guiar el desarrollo de sistemas de software. A continuación, se presentan estos principios:

- Rigor y formalidad: La documentación ambigua o inconsistente dificulta la detección de errores y los cambios necesarios en el sistema. Al aplicar rigor y formalidad en la documentación y el código, se incrementa la fiabilidad, verificabilidad y mantenibilidad del sistema.
- Modularidad: Dividir un sistema complejo en componentes más pequeños facilita el diseño, desarrollo, prueba y modificación de cada parte de manera independiente. En la programación estructurada, los módulos suelen ser procedimientos y funciones que proporcionan funcionalidad al sistema; en la programación orientada a objetos, los módulos son clases que representan elementos del problema.

- Abstracción: Abstraer implica identificar la esencia del problema, extrayendo características comunes de ejemplos específicos. Por ejemplo, de un conjunto de figuras geométricas se puede abstraer la idea de que todas son elipses, independientemente de sus variaciones en tamaño o forma.
- Anticipación al cambio: El software está sujeto a constantes cambios, ya sea para corregir defectos, adaptar nuevos requerimientos o modificar los existentes. Es importante prever en qué partes del sistema es probable que ocurran cambios y gestionar las diferentes versiones mediante la configuración del software.
- Generalidad: En lugar de crear múltiples soluciones específicas, se busca resolver problemas de manera general, permitiendo la reutilización de módulos desde varios puntos de la aplicación y evitando soluciones redundantes.
- Incrementalidad: Durante el desarrollo del software, los requerimientos del usuario tienden a redefinirse o ajustarse. Por ello, la entrega incremental permite entregar prototipos o versiones parciales del sistema, obteniendo retroalimentación temprana y afinando los requerimientos reales del usuario. Se puede comenzar con un núcleo funcional y luego añadir más características en fases posteriores.
- Separación de intereses: Este principio implica dividir diferentes aspectos del problema para abordarlos de forma independiente. Por ejemplo, se puede desarrollar primero una solución correcta y luego mejorarla en eficiencia, de forma secuencial.

3.2.4. ¿CUÁL ES EL TRABAJO DE LOS INGENIEROS DE SOFTWARE?

Los ingenieros de software se centran en el desarrollo de productos de software destinados a la venta para un cliente. Hay dos tipos principales de productos de software:

Productos genéricos.

Son sistemas independientes creados por una empresa de desarrollo y comercializados en el mercado abierto, de modo que cualquier cliente interesado puede adquirirlos. Ejemplos de este tipo incluyen software para PCs, como bases de datos, procesadores de texto, programas de diseño y herramientas para la gestión de proyectos.

Productos personalizados o a medida.

Estos sistemas son solicitados específicamente por un cliente en particular, y una empresa de software los desarrolla de acuerdo con las necesidades exclusivas de ese cliente. Ejemplos de software a medida incluyen sistemas de control para dispositivos electrónicos, software para gestionar procesos de negocio específicos y sistemas de control del tráfico aéreo.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMAGEN 7. Ingeniero de software realizando su trabajo.

Una diferencia clave entre estos tipos de software es que, en los productos genéricos, la empresa que desarrolla el software define sus especificaciones. En cambio, en los productos personalizados, normalmente es la organización que contrata el software quien establece y controla sus especificaciones, y los desarrolladores deben trabajar conforme a ellas. Sin embargo, la línea entre estos dos tipos de productos se está volviendo cada vez más difusa. Muchas empresas de software ahora crean un sistema genérico que luego personalizan para ajustarlo a las necesidades de un cliente específico. Los sistemas de planificación de recursos empresariales (ERP), como los sistemas SAP, ejemplifican bien este enfoque, ya que un sistema extenso y complejo se adapta a una empresa integrando reglas de negocio, procesos, informes, y más.

3.2.5. ¿CÓMO SE EVALUA LA CALIDAD DE UN PRODUCTO DE SOFTWARE?

La calidad de un producto puede evaluarse desde múltiples perspectivas, y el software no es la excepción, ya que existen diferentes enfoques para medir su calidad. Idealmente, podríamos evaluar la calidad del software con precisión, de forma similar a cómo se miden aspectos en otros productos de ingeniería, como la resistencia de materiales o las dimensiones exactas. Sin embargo, la evaluación de la calidad del software no es tan sencilla, y las técnicas para aplicar métricas precisas están en constante desarrollo. McCall y otros propusieron un esquema general para evaluar la calidad del software, estructurado en tres niveles: factores, criterios y métricas. Los factores de calidad representan el nivel más alto y son la evaluación global de la calidad. Sin embargo, esta evaluación no se hace directamente, sino a través de criterios intermedios que influyen en los factores de calidad. Las métricas, situadas en el nivel inferior, son mediciones concretas de características específicas del producto, y constituyen la base para evaluar los criterios intermedios. Entre los factores de calidad que se destacan, tenemos:

- **Corrección:** Mide qué tan bien el software cumple con sus especificaciones, a menudo evaluado por el porcentaje de requisitos cumplidos.
- **Fiabilidad:** Refleja la ausencia de fallos durante el uso del software, estimada por el número de fallos o el tiempo de inactividad en un período específico.
- **Eficiencia:** Mide la relación entre los resultados obtenidos y los recursos utilizados, generalmente como la inversa de los recursos consumidos para realizar una operación.
- **Seguridad:** Se refiere a la dificultad de acceso no autorizado a los datos o funciones.
- **Facilidad de uso:** Mide el esfuerzo requerido para aprender a usar el software y usarlo correctamente.
- **Mantenibilidad:** Es la facilidad para corregir problemas en el software, principalmente en el mantenimiento correctivo.
- **Flexibilidad:** Mide la facilidad para modificar el software, especialmente durante el mantenimiento adaptativo y perfectivo.
- **Facilidad de prueba:** Se refiere a la facilidad para probar el software y verificar su corrección o fiabilidad.

- **Transportabilidad:** Mide la facilidad para adaptar el software a otra plataforma distinta de la original.
- **Reusabilidad:** Evalúa la facilidad para reutilizar partes del software en futuros desarrollos, facilitada por una buena organización de módulos y funciones.
- **Interoperabilidad:** mide la capacidad del software para operar en combinación con otros productos.

Comprobar la calidad de un software es una tarea que implica evaluar su funcionamiento en condiciones específicas para detectar posibles errores. Aunque las pruebas buscan garantizar que el software cumple ciertos criterios de calidad, no aseguran la ausencia total de errores. Si se detecta un error, se confirma que el producto tiene fallas, pero él no encontrar errores no significa que no existan, ya que el software no puede evaluarse exhaustivamente y solo se prueba una pequeña fracción de sus posibles funciones. El desarrollo y mantenimiento de software presentan desafíos únicos en comparación con la manufactura de hardware. Aunque en ambos casos se persigue una alta calidad a través de un buen diseño, el software no enfrenta problemas de calidad en la fase de producción, como ocurre con el hardware. Además, mientras que el hardware suele desgastarse y fallar con el tiempo por factores ambientales, el software no se "desgasta" en el mismo sentido.

La "curva de tina" del hardware (ver figura 4) muestra una alta tasa de fallos al inicio de su vida útil, que disminuye tras resolver defectos de fabricación y luego aumenta con el desgaste. En contraste, la "curva idealizada" del software (ver figura 5) indica una estabilización de errores tras su corrección inicial. Sin embargo, el software sí puede "deteriorarse" debido a los cambios y actualizaciones que se le realizan a lo largo del tiempo, lo cual aumenta la complejidad de su mantenimiento y provoca picos en la tasa de fallos. A diferencia del hardware, el software no permite reemplazar componentes defectuosos; cada falla en software se origina en un error en el diseño o en el proceso de codificación. Además, aunque el uso de componentes reutilizables está avanzando, la mayoría del software aún se construye con un propósito específico. En disciplinas como la ingeniería mecánica o eléctrica, el uso de componentes estandarizados, como tornillos y circuitos, facilita el diseño. Para el software, este enfoque es relativamente nuevo y plantea la necesidad de diseñar componentes que sean reutilizables en distintos programas.

Estos componentes modernos incluyen tanto datos como los procesos que los manipulan, permitiendo construir aplicaciones mediante la combinación de partes reutilizables, como librerías de interfaces de usuario con ventanas, menús y otros elementos interactivos.

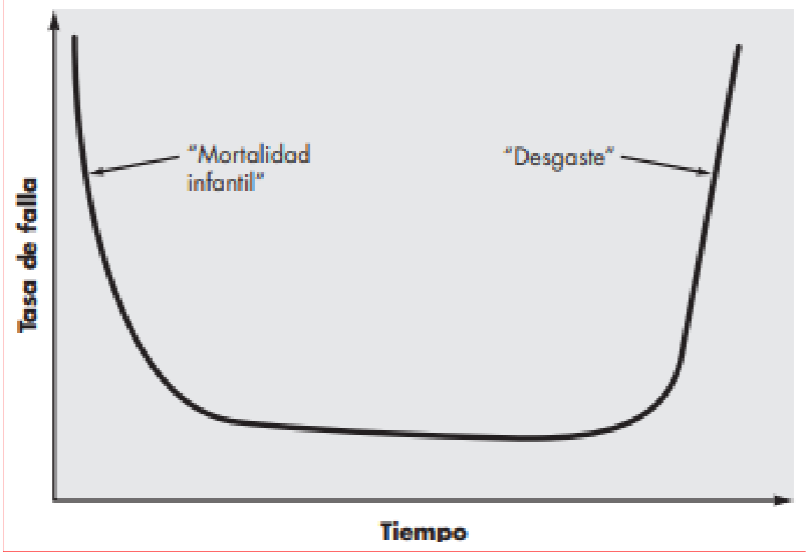

Figura 4: "Curva de tina" del hardware que muestra una alta tasa de fallos.

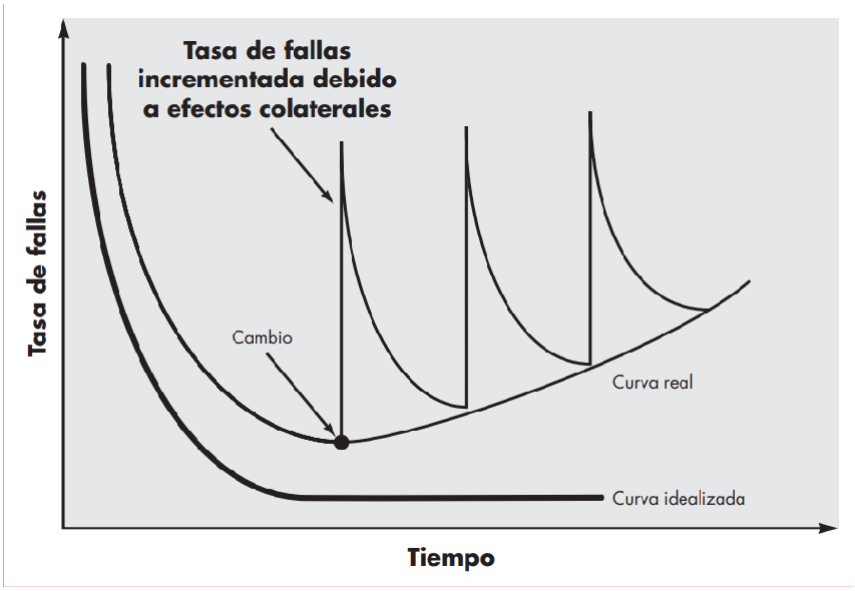

Figura 5: "Curva idealizada" del software que muestra una estabilización y la posibilidad de deterioro.

3.2.6. EN COMPARACIÓN CON OTRAS ÁREAS

Naturaleza Intangible del Producto y Relación con Otras Disciplinas de Ingeniería.

A diferencia de otras ingenierías, como la ingeniería civil o la ingeniería eléctrica, cuyo enfoque principal es la creación de productos físicos (edificios, puentes, redes eléctricas, etc.), la ingeniería de software lidia con un producto intangible: el software. Este aspecto fundamental implica que, mientras en las ingenierías tradicionales se utilizan materiales físicos y procesos de manufactura, en la ingeniería de software la "materia prima" es el conocimiento. El desarrollo y mantenimiento del software se centran en gestionar información y procesos más que en manipular materiales tangibles, lo cual añade una capa de complejidad en la conceptualización y gestión de cambios durante el desarrollo. Aunque el software es relativamente fácil de modificar, su naturaleza intangible dificulta su visualización, lo que añade complejidad a su gestión y desarrollo (Sánchez et al., 2012, pp. 10-12).

Producción y Manufactura

En las ingenierías tradicionales, existe un proceso de manufactura donde cada unidad producida puede ser distinta y sufre desgaste físico con el tiempo. En cambio, el software, una vez desarrollado, se puede replicar de manera exacta y a bajo costo, sin necesidad de procesos de manufactura adicionales ni riesgo de desgaste físico. Sin embargo, a pesar de no experimentar desgaste tangible, el software es susceptible a la obsolescencia, ya que las demandas tecnológicas y del mercado evolucionan rápidamente, exigiendo actualizaciones y mejoras constantes para mantenerse relevante.

Enfoque Científico en la Ingeniería de Software

La Ingeniería de Software no solo se centra en la creación de productos funcionales, sino también en el desarrollo de teorías y modelos que permiten comprender y mejorar los procesos de desarrollo, lo que le confiere un enfoque científico. Esta área colinda con ciencias de la computación, matemáticas y física, debido a su énfasis en optimizar y estudiar la complejidad del software y en investigar métodos para abordarla. Por ejemplo, teorías como las "Leyes de la Evolución del Software" de Lehman ayudan a explicar fenómenos recurrentes en el mantenimiento y evolución del software (Sánchez et al., 2012, p. 11). Aun cuando la ingeniería de software toma metodologías y conceptos de otras ingenierías, estos deben adaptarse a la naturaleza intangible del software, manteniéndose así dentro de un enfoque científico que optimice su desarrollo.

Mantenimiento y Evolución

Aunque el software no se desgasta físicamente como otros productos de ingeniería (por ejemplo, máquinas o infraestructuras), puede "deteriorarse" de manera conceptual debido a la acumulación de fallos o a la falta de compatibilidad con nuevos entornos operativos. En este sentido, el mantenimiento del software presenta un desafío distinto, ya que no se limita a reparaciones físicas, sino que requiere revisiones y actualizaciones continuas en el diseño y funcionalidad. A medida que el software se mantiene y evoluciona, el proceso requiere una gestión meticulosa para identificar y resolver errores o adaptarlo a nuevos requisitos.

Adaptación de Conocimientos y Metodologías de Otras Ingenierías

A pesar de los desafíos únicos de la ingeniería de software, esta se ha beneficiado de principios y metodologías de otras ingenierías. Por ejemplo, la gestión de proyectos en ingeniería civil y el modelado estructurado se han adaptado para cumplir con los requerimientos específicos del desarrollo de software, permitiendo una mayor eficiencia en la planificación y en el control de cambios. Sin embargo, existen límites en esta extrapolación, ya que la manufactura y el diseño físico que caracterizan a otras ingenierías no se trasladan de forma directa al software. Por lo tanto, la ingeniería de software se enfrenta a la necesidad de adaptar y personalizar las prácticas de otras ingenierías a su contexto intangible y en constante evolución.

Interacción con Otras Disciplinas de la Computación y Ciencias

La ingeniería de software también comparte ciertos aspectos con otras áreas como la ciencia de la computación y la ingeniería de sistemas. A diferencia de la ciencia de la computación, que se enfoca en teorías y algoritmos, la ingeniería de software los aplica en un contexto práctico y orientado a la creación de productos funcionales. En cuanto a su relación con la ingeniería de sistemas, esta última abarca un enfoque más amplio que integra tanto hardware como software, mientras que la ingeniería de software se centra en los componentes de software en sistemas complejos.

La Ingeniería de Software como Profesión

La Ingeniería de Software ha alcanzado el estatus de disciplina profesional, respaldada por la consolidación de programas curriculares y por la creación de organizaciones acreditadas, como ACM e IEEE, que desarrollan y promueven estándares de calidad. Esto marca una diferencia notable en comparación con sus inicios, cuando era común que los programadores trabajaran de forma autodidacta. Actualmente, se exige un perfil profesional, validado por competencias reguladas dentro de una estructura organizativa y orientado a

objetivos específicos, lo cual posiciona la ingeniería de software dentro de los estándares éticos y profesionales de otras ingenierías (Sánchez et al., 2012, pp. 16-18).

Obsolescencia y Rápido Ciclo de Vida

Uno de los desafíos más característicos de la ingeniería de software en comparación con otras ingenierías es el corto ciclo de vida de sus productos. Los avances tecnológicos, junto con las cambiantes demandas del mercado y los usuarios, obligan a que el software se desarrolle de manera ágil y eficiente, pues su relevancia es más breve en comparación con productos como los vehículos o las infraestructuras, que poseen un ciclo de vida más prolongado.

4. EJEMPLOS

4.1. ASPECTOS GENERALES DEL SOFTWARE

4.1.1. EJEMPLO 1

¿Cómo define el IEEE el software y cuáles son los elementos que incluye esta definición en relación con las operaciones de un sistema de cómputo?

PASO 1.

El IEEE define el software como "el conjunto de los programas de computación, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo". Esta definición abarca no solo los programas que indican las instrucciones que el sistema debe ejecutar, sino también los procedimientos, normas y toda la documentación y datos que están involucrados en el funcionamiento de dicho sistema.

4.1.2. EJEMPLO 2

¿Cuáles son los tres componentes principales del software y cómo contribuyen cada uno al funcionamiento de un sistema de software, considerando tanto los programas como los elementos adicionales que aseguran su operatividad?

PASO 1.

El software se compone principalmente de tres elementos:

PASO 2.

Instrucciones o programas, que al ejecutarse proporcionan las características, funciones y rendimiento deseados, convirtiéndose en el núcleo operativo del sistema.

PASO 3.

Estructuras de datos, que facilitan la manipulación eficiente de la información y son esenciales para la gestión de datos dentro del sistema.

PASO 4.

Información explicativa, que se presenta en formatos impresos y digitales para detallar el funcionamiento y el uso de los programas. Además, el software no solo abarca los programas, sino también toda la documentación y configuración de datos necesarios para que estos programas funcionen adecuadamente. Esto incluye programas independientes, archivos de configuración, sistemas de documentación que describen la estructura y el uso del sistema, guías de usuario, y sitios web de soporte donde los usuarios pueden acceder a actualizaciones o información sobre nuevos productos.

4.1.3. EJEMPLO 3

¿Cómo clasifica Roger Pressman el software y cuáles son las características distintivas de cada tipo, especialmente en relación con su propósito y entorno de aplicación?

PASO 1.

Roger Pressman clasifica el software en siete categorías principales, cada una diseñada para diferentes propósitos y entornos de aplicación:

PASO 2.

Software de sistemas: Actúa como la base que permite la ejecución de otros programas, incluyendo sistemas operativos y compiladores, con una estrecha interacción con el hardware y un enfoque en la eficiencia.

PASO 3.

Software de aplicación: Soluciones específicas para satisfacer necesidades de organizaciones o individuos, como sistemas de gestión empresarial y ERP.

PASO 4.

Software científico y de ingeniería: Se especializa en cálculos complejos y simulaciones avanzadas, usado en investigación científica y simulaciones de sistemas complejos.

PASO 5.

Software incrustado o empotrado: Integrado en dispositivos físicos, controla su funcionamiento en tiempo real, como en electrodomésticos y sistemas automotrices.

PASO 6.

Software de línea de productos: Software orientado al consumidor para necesidades generales, como hojas de cálculo y procesadores de texto.

PASO 7.

Aplicaciones web: Webapps que integran bases de datos y servicios distribuidos para operaciones en línea, como banca y compras, caracterizadas por su accesibilidad y rapidez.

PASO 8.

Software de inteligencia artificial: Incluye técnicas avanzadas para resolver problemas complejos, como en sistemas expertos y redes neuronales, permitiendo aprendizaje y adaptación en diferentes escenarios.

4.1.4. EJEMPLO 4

¿Cuáles son las principales ventajas, desventajas y desafíos del software en la economía moderna, y cómo impactan estos elementos en la responsabilidad de los desarrolladores y en el ciclo de vida del software?

PASO 1.

Las ventajas principales del software incluyen la transformación de modelos de negocio (como plataformas de transporte y ventas en línea), el impulso de tecnologías innovadoras (por ejemplo, en ingeniería genética), y la expansión de tecnologías de comunicación. Además, el software ha digitalizado contenidos, impulsado la inteligencia artificial y análisis de datos, y ha mejorado la disponibilidad

y accesibilidad de productos tecnológicos. También ha generado gran impacto económico, ya que muchas de las mayores empresas del mundo son de software, y su doble valor radica en ser tanto un producto como una herramienta para gestionar información valiosa.

PASO 2.

Las desventajas se centran en el impacto que pueden tener los fallos del software en sectores cruciales como el bancario y gubernamental, y la gran responsabilidad que recae en los desarrolladores para garantizar productos seguros y confiables.

PASO 3.

Los desafíos del desarrollo de software incluyen tiempos de desarrollo prolongados, dificultades para estimar costos y plazos, la complejidad en medir el progreso debido a la intangibilidad del software, y los altos costos tanto de desarrollo como de mantenimiento (que representan hasta el 80% del esfuerzo). Además, existen expectativas de tiempos de entrega cortos por parte de los clientes, lo cual genera presión adicional sobre el proceso de desarrollo y entrega de productos de software.

4.2. INGENIERÍA DE SOFTWARE

4.2.1. EJEMPLO 5

¿Qué es la ingeniería de software y cuáles son sus capas esenciales según la perspectiva de distintos autores?

PASO 1.

La ingeniería de software es una disciplina de ingeniería que abarca el diseño, desarrollo y mantenimiento de sistemas de software de manera sistemática y rentable, asegurando que estos cumplan con los requisitos del cliente. A diferencia de simplemente programar, este campo incluye la planificación, gestión de proyectos y mantenimiento del software a lo largo de su vida útil. La ingeniería de software tiene como objetivo crear sistemas fiables y eficientes mediante el uso de métodos técnicos y herramientas, respetando limitaciones financieras y organizativas.
PASO 2.

Varios autores han definido esta disciplina de manera similar. Bauer (1972) destaca el uso de principios sólidos de ingeniería para producir software confiable. Zelkovitz (1978) la define como el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas de software. Sommerville (2004) expone que es una disciplina que abarca desde la especificación del sistema hasta su mantenimiento, y Pressman (2005) la describe como la integración de procesos, métodos y herramientas para el desarrollo de software.

PASO 3.

La ingeniería de software se estructura en varias capas esenciales. La base es un compromiso organizacional con la calidad, impulsado por filosofías de mejora continua como Six Sigma. Sobre esta base se encuentra la capa de proceso, que coordina las capas tecnológicas y establece una estructura lógica para gestionar proyectos, asegurar la calidad y gestionar el cambio. Los métodos de ingeniería de software aportan el conocimiento técnico necesario para desarrollar el software, abarcando desde la comunicación y análisis de requisitos hasta las pruebas y el soporte. Por último, las herramientas proporcionan soporte automatizado o semiautomatizado, facilitando el desarrollo a través de entornos integrados conocidos como ingeniería de software asistida por computadora.

PASO 4.

Según Bruegge y Dutoit (2004), la ingeniería de software también implica a participantes diversos, conocidos como "partes interesadas", desde programadores hasta directivos y clientes, ya que el éxito del proyecto depende de la contribución de diversas habilidades.

4.2.2. EJEMPLO 6

¿Cómo se integran los principios de gestión de recursos y actividades sombrilla en el proceso de desarrollo de software para garantizar la calidad y eficiencia del producto final, y cuáles son las implicaciones de una mala gestión en estas áreas?

PASO 1.

La integración de los principios de gestión de recursos y actividades sombrilla es crucial en el proceso de desarrollo de software, ya que ambas áreas son fundamentales para asegurar que el proyecto se desarrolle de manera fluida, eficiente y con un enfoque en la calidad. La gestión de recursos abarca la administración efectiva del tiempo, el presupuesto, el personal y el equipamiento necesario a lo largo del ciclo de vida del proyecto. Por otro lado, las actividades sombrilla, que incluyen el seguimiento y control del proyecto, la gestión de

riesgos y el aseguramiento de la calidad, ofrecen un marco estructurado que permite al equipo de desarrollo evaluar el progreso y la calidad de manera continua.

PASO 2.

Una buena gestión de recursos implica asignar adecuadamente el personal y los presupuestos a cada fase del proyecto, lo que es esencial para cumplir con los plazos y los estándares de calidad. Cuando la gestión de recursos es efectiva, se pueden realizar ajustes en tiempo real en función del progreso del proyecto y de los cambios en las necesidades del cliente, lo que contribuye a una mejor alineación entre las expectativas del cliente y los resultados del proyecto.

PASO 3.

Por otro lado, las actividades sombrilla permiten una supervisión constante y la implementación de mejoras incrementales. Por ejemplo, el seguimiento y control del proyecto permite identificar desviaciones del plan original, lo que ayuda a aplicar acciones correctivas antes de que se conviertan en problemas mayores. Asimismo, la gestión de riesgos ayuda a anticipar y mitigar posibles problemas que podrían impactar el desarrollo o la calidad del software.

PASO

4.

Las implicaciones de una mala gestión en estas áreas pueden ser significativas. Si los recursos no se gestionan adecuadamente, se corre el riesgo de exceder el presupuesto, de no cumplir con los plazos establecidos y de provocar una sobrecarga en el equipo de trabajo, lo que podría llevar a la desmotivación y a un incremento en la tasa de errores. En cuanto a las actividades sombrilla, una falta de seguimiento y control puede resultar en un software que no cumple con los estándares de calidad, con errores no detectados en fases anteriores que se manifiestan en el producto final. Esto no solo afecta la satisfacción del cliente, sino que también puede conllevar costos adicionales para corregir errores y mejorar el producto después de su entrega.

4.2.3. EJEMPLO 7

¿Cuál es la importancia de los principios de rigor y formalidad, modularidad y anticipación al cambio en la ingeniería de software, y cómo pueden influir en el ciclo de vida del desarrollo de un sistema?

PASO 1.

Los principios de rigor y formalidad, modularidad y anticipación al cambio son fundamentales en la ingeniería de software, ya que cada uno de ellos aborda aspectos críticos del desarrollo de sistemas que impactan en la calidad, mantenibilidad y adaptabilidad del software a lo largo de su ciclo de vida.

PASO 2.

Rigor y Formalidad: Este principio subraya la importancia de mantener documentación clara y precisa, así como código que siga estándares establecidos. La documentación ambigua puede generar confusiones y errores que son difíciles de detectar y corregir. La aplicación de rigor y formalidad mejora la fiabilidad y verificabilidad del sistema, ya que permite a los desarrolladores y a otros interesados comprender claramente el funcionamiento y las expectativas del software. Esto es

PASO

especialmente crucial en etapas de prueba y mantenimiento, donde la identificación rápida de errores puede evitar costos adicionales y retrasos en el proyecto.

3.

Modularidad: Dividir un sistema complejo en módulos más pequeños permite un enfoque más manejable en el diseño, desarrollo, prueba y mantenimiento. Cada módulo puede ser desarrollado y probado de manera independiente, lo que no solo facilita la colaboración entre equipos de trabajo, sino que también mejora la mantenibilidad del sistema a largo plazo. Si se requiere una modificación, se puede hacer en un módulo específico sin afectar el resto del sistema, lo que minimiza el riesgo de introducir nuevos errores y reduce el tiempo de inactividad. Además, la modularidad fomenta la reutilización de componentes, lo que puede resultar en un ahorro significativo de tiempo y recursos en futuros proyectos.

PASO 4.

Anticipación al Cambio: En el dinámico entorno del desarrollo de software, los cambios son inevitables. Este principio enfatiza la necesidad de prever y gestionar cambios en los requerimientos a lo largo del ciclo de vida del software. La capacidad de anticiparse a los cambios y adaptarse rápidamente permite que el equipo de desarrollo mantenga la relevancia del sistema en relación con las necesidades del cliente y el entorno tecnológico. La gestión adecuada de la configuración del software y la planificación de la arquitectura del sistema con flexibilidad en mente son cruciales para evitar sobrecostos y asegurar que el software pueda evolucionar conforme a las demandas del mercado.

4.2.4. EJEMPLO 8

¿Cuáles son las diferencias fundamentales entre el desarrollo de productos de software genéricos y personalizados, y cómo impacta cada enfoque en el proceso de desarrollo y en la relación con los clientes?

PASO 1.

PASO

El desarrollo de productos de software genéricos y personalizados presenta diferencias fundamentales en su concepción, proceso de desarrollo y la relación que establecen con los clientes. Estos enfoques, aunque a menudo se entrelazan en la práctica, tienen implicaciones significativas en cómo se lleva a cabo el trabajo de los ingenieros de software.

2.

Especificaciones y Control: En el caso de los productos genéricos, la empresa de desarrollo define las especificaciones del software, lo que significa que tiene control total sobre el diseño y las características del producto. Esto permite a los desarrolladores centrarse en crear una solución que pueda ser utilizada por una amplia variedad de clientes, optimizando así el proceso de desarrollo para atender necesidades comunes. Por otro lado, en los productos personalizados, son los clientes quienes establecen las especificaciones, y los ingenieros deben adaptar su trabajo para cumplir con estos requisitos específicos. Este enfoque puede ser más desafiante, ya que requiere una comprensión profunda de las necesidades del cliente y una flexibilidad en el proceso de desarrollo.

PASO 3.

Adaptabilidad y Flexibilidad: La tendencia hacia la creación de productos que son inicialmente genéricos pero que luego se personalizan para un cliente específico está en aumento. Este enfoque híbrido permite a las empresas de software ofrecer soluciones más adaptadas sin partir de cero. Por ejemplo, los sistemas de planificación de recursos empresariales (ERP), como los sistemas SAP, son diseñados como productos genéricos que pueden ser ajustados a las necesidades específicas de cada organización. Esto combina las ventajas de un producto ampliamente probado con la personalización necesaria para adaptarse a procesos y requerimientos únicos. Relación con el Cliente: En el desarrollo de productos personalizados, la relación entre los ingenieros de software y el cliente tiende a ser más estrecha. Los desarrolladores deben trabajar en colaboración con el cliente para comprender y definir los requisitos, lo que puede fomentar una comunicación más fluida y continua. En contraste, en el desarrollo de productos genéricos, la interacción con los clientes

PASO

puede ser menos directa, ya que el producto se destina a un mercado más amplio y se basa en las necesidades de múltiples usuarios.

PASO 4.

Impacto en el Proceso de Desarrollo: El desarrollo de productos genéricos suele ser más eficiente en términos de tiempo y costos, ya que una vez que se define el producto, puede ser replicado y vendido a varios clientes sin necesidad de personalización. En cambio, el desarrollo de productos personalizados puede ser más prolongado y costoso, debido a la necesidad de iteraciones y ajustes continuos basados en los

comentarios del cliente. Sin embargo, los productos personalizados pueden ofrecer un mayor valor al cliente al satisfacer exactamente sus necesidades, lo que puede resultar en una mayor satisfacción y lealtad.

4.2.5. EJEMPLO 9

¿Cómo se evalúa la calidad de un producto de software y cuáles son los factores clave que se consideran en este proceso?

PASO 1.

La evaluación de la calidad de un producto de software es un proceso complejo que se aborda desde múltiples perspectivas y no se limita a una única métrica. Al igual que en otros productos de ingeniería, se busca medir la calidad de manera precisa, pero esto se complica debido a la naturaleza del software. McCall y otros investigadores han propuesto un esquema general que categoriza la evaluación de la calidad del software en tres niveles: factores, criterios y métricas. A continuación, se detallan estos elementos y los factores clave que se consideran en la evaluación de la calidad del software.

PASO 2.

Estructura de Evaluación:

PASO 3.

Factores de Calidad: Son el nivel más alto de evaluación y representan una visión global de la calidad del software. Estos factores no se evalúan directamente, sino a través de criterios intermedios que influyen en ellos.

PASO 4.

Criterios Intermedios: Estos criterios sirven como indicadores que permiten medir los factores de calidad.

PASO 5.

Métricas: En el nivel más bajo, las métricas son mediciones concretas de características específicas del software, y forman la base sobre la que se evalúan los criterios intermedios.

PASO 6.

Factores Clave de Calidad:

PASO 7.

Corrección: Mide cómo se ajusta el software a sus especificaciones, a menudo evaluado por el porcentaje de requisitos cumplidos.

PASO 8.

Fiabilidad: Refleja la ausencia de fallos durante el uso del software, estimándose por el número de fallos o el tiempo de inactividad en un período específico.

PASO 9.

Eficiencia: Mide la relación entre los resultados obtenidos y los recursos utilizados, generalmente como la inversa de los recursos consumidos para realizar una operación.

PASO 10.

Seguridad: Se refiere a la dificultad de acceso no autorizado a los datos o funciones del software.

PASO 11.

Facilidad de Uso: Evalúa el esfuerzo necesario para aprender a usar el software y utilizarlo correctamente.

PASO 5.

Mantenibilidad: Refleja la facilidad con la que se pueden corregir problemas en el software, particularmente en el mantenimiento correctivo.

PASO 13.

Flexibilidad: Mide la facilidad para modificar el software, especialmente durante el mantenimiento adaptativo y perfectivo.

PASO 14.

Facilidad de Prueba: Indica la facilidad con la que se puede probar el software y verificar su corrección o fiabilidad.

PASO 15.

Transportabilidad: Mide la facilidad para adaptar el software a otra plataforma diferente de la original.

PASO 16.

Reusabilidad: Evalúa la facilidad de reutilizar partes del software en futuros desarrollos, facilitada por una buena organización de módulos y funciones. Interoperabilidad: Mide la capacidad del software para operar en conjunto con otros productos.

PASO 17.

Desafíos en la Evaluación: La comprobación de la calidad del software implica evaluar su funcionamiento en condiciones específicas para detectar errores. A pesar de las pruebas realizadas, no se puede garantizar la ausencia total de errores, ya que el software es complejo y solo se prueba una pequeña fracción de sus posibles funciones. Además, el software no enfrenta problemas de calidad en la fase de producción como ocurre con el hardware, y aunque el hardware tiende a desgastarse con el tiempo, el software puede deteriorarse debido a cambios y actualizaciones que aumentan su complejidad.

PASO 18.

Curvas de Calidad: En comparación con el hardware, el software presenta diferentes patrones de calidad a lo largo de su ciclo de vida. La "curva de tina" del hardware muestra una alta tasa de fallos al inicio de su vida útil, que luego disminuye, mientras que la "curva idealizada" del software indica una estabilización de errores tras las correcciones iniciales. Sin embargo, el software puede "deteriorarse" con el tiempo debido a cambios, lo que puede generar picos en la tasa de fallos.

PASO 19.

Componentes Reutilizables: Aunque el uso de componentes reutilizables está en aumento, la mayoría del software todavía se desarrolla con un propósito específico. A diferencia de disciplinas como la ingeniería mecánica, donde se utilizan componentes estandarizados, la reutilización en software plantea la necesidad de diseñar componentes que puedan ser utilizados en diferentes programas, lo que permite construir aplicaciones combinando partes reutilizables, como librerías de interfaces de usuario.

4.2.6. EJEMPLO 10

¿Cómo se diferencia la ingeniería de software de otras disciplinas de ingeniería en términos de su naturaleza intangible, el enfoque en la producción y el mantenimiento, y cómo estas diferencias afectan la aplicación de metodologías de gestión de proyectos?

PASO 1.

La ingeniería de software se diferencia de otras disciplinas de ingeniería, como la ingeniería civil o eléctrica, principalmente por su naturaleza intangible. A diferencia de las ingenierías tradicionales que producen bienes físicos, el software es un producto abstracto cuya "materia prima" es el conocimiento. Esto implica que, aunque el software sea más fácil de modificar en comparación con los productos físicos, su falta de tangibilidad complica la visualización y gestión de cambios durante su desarrollo, aumentando la complejidad del proceso (Sánchez et al., 2012).

PASO 2.

En cuanto a la producción y el mantenimiento, la ingeniería de software no tiene un proceso de manufactura que implique desgaste físico; el software puede replicarse de manera exacta y a bajo costo una vez desarrollado. Sin embargo, es susceptible a la obsolescencia debido a las rápidas evoluciones tecnológicas y del mercado, lo que requiere actualizaciones constantes. Por otro lado, aunque el software no se "desgasta" como el hardware, puede "deteriorarse" conceptualmente por la acumulación de fallos o incompatibilidades con nuevos entornos operativos, lo que convierte el mantenimiento en un desafío continuo que implica revisiones y adaptaciones en el diseño y la funcionalidad.

PASO 3.

Estas diferencias fundamentales afectan la aplicación de metodologías de gestión de proyectos. Mientras que principios de otras ingenierías, como la gestión de proyectos en ingeniería civil, pueden adaptarse al desarrollo de software, la naturaleza intangible del software requiere personalizar y adaptar estas prácticas para abordar las particularidades de su entorno en constante cambio. La ingeniería de software, además, se centra en la creación de productos funcionales a partir de teorías y algoritmos desarrollados en la ciencia de la computación, lo que contrasta con otras disciplinas que a menudo se enfocan en el diseño y manufactura física de productos.

5. CUESTIONARIO

1. ¿Cómo define el IEEE el software?

A) Como el conjunto de programas que especifican las instrucciones que debe seguir el sistema.

B) Como la documentación y procedimientos necesarios para el desarrollo de hardware. **C) Como el conjunto de programas, procedimientos, reglas, documentación y datos asociados de un sistema de cómputo.**

D) Como el conjunto de aplicaciones orientadas al usuario.

Referencia: Primer párrafo, definición formal de software según el IEEE.

---

2. ¿Cuál de las siguientes es una característica principal del software de sistemas?

A) Está diseñado para calcular simulaciones avanzadas.

B) Proporciona funcionalidad específica y general para consumidores.

C) **Optimiza el acceso y la eficiencia de otros programas y usuarios.**

D) Especifica las reglas de negocio en aplicaciones empresariales.

Referencia: Sección sobre clasificación de software, categoría "Software de sistemas".

---

3. ¿Cuál de las siguientes es una ventaja del software en la vida cotidiana?

A) Dificultad para medir el progreso.
B) **Expansión de tecnologías de comunicación.**
C) Costos de desarrollo elevados.
D) Complejidad en las estimaciones de tiempo.

Referencia: Ventajas del software en nuestra vida cotidiana, punto sobre la expansión de tecnologías existentes.

---

4. ¿Qué tipo de software se utiliza para cálculos complejos y simulaciones en áreas científicas?

A) Software de sistemas.
B) Aplicaciones web.
C) **Software científico y de ingeniería.**
D) Software de línea de productos.

Referencia: Sección de clasificación de software, categoría "Software científico y de ingeniería".

---

5. ¿Cuál es una de las principales desventajas del software?

A) La imposibilidad de realizar pruebas para verificar su funcionamiento.
B) El reemplazo constante de componentes defectuosos.
C) **Los altos costos de mantenimiento.**
D) Su corta vida útil comparado con el hardware.

Referencia: Desventajas del software y sus desafíos, punto sobre elevados costos de mantenimiento.

---

6. ¿Qué mide la mantenibilidad como factor de calidad del software?

A) La facilidad para modificar el software.
B) La ausencia de fallos durante el uso.
C) La capacidad para adaptarse a plataformas distintas.
D) **La facilidad para corregir problemas en el software.**

Referencia: Factores de calidad del software, mantenibilidad.

---

7. ¿Cómo se define un producto de software genérico?

A) Un software solicitado por un cliente específico.
B) Un software desarrollado a medida para una organización.
C) **Un sistema desarrollado y comercializado en el mercado abierto.**
D) Un sistema personalizado para sistemas empresariales.

Referencia: Sección sobre productos de software, definición de productos genéricos.

---

8. ¿Qué implica la corrección como factor de calidad en el software?

A) Medir la eficiencia de los recursos utilizados.
B) **Determinar si cumple con sus especificaciones.**
C) Evaluar el esfuerzo para aprender a usar el software.
D) Determinar la capacidad de interoperabilidad.

Referencia: Factores de calidad del software, corrección.

---

9. ¿Cuál es un ejemplo de software incrustado?

A) Un sistema operativo de uso general.

B) Un programa de diseño gráfico.
C) **El software de control en un electrodoméstico.**
D) Un software ERP para empresas.

Referencia: Sección de clasificación de software, categoría "Software incrustado o empotrado".

---

10. ¿Qué mide la facilidad de prueba en el software?

A) El esfuerzo para modificar el software.
B) **La facilidad para verificar su corrección o fiabilidad.**
C) La posibilidad de adaptarlo a otras plataformas.
D) La facilidad para documentar los cambios.

Referencia: Factores de calidad del software, facilidad de prueba.

---

11. ¿Qué caracteriza al software de aplicación?

A) Es un software orientado exclusivamente a desarrolladores.
B) Se enfoca en optimizar el acceso y la eficiencia de recursos.
C) **Atiende necesidades específicas de individuos u organizaciones.**
D) Es utilizado principalmente en simulaciones científicas.

Referencia: Sección de clasificación de software, categoría "Software de aplicación".

---

12. ¿Cuál es un desafío común en el desarrollo de software?

A) La facilidad para medir su calidad de manera directa.
B) La interacción constante con componentes de hardware.
C) La disponibilidad de componentes estandarizados.
D) **La capacidad de probar todas las funciones del software.**

Referencia: Desventajas del software y sus desafíos, sobre la dificultad de garantizar calidad.

---

13. ¿Cuál es un ejemplo de una aplicación web?

A) Un sistema operativo.
B) Un programa de edición de imágenes.
C) **Una aplicación bancaria en línea.**
D) Un programa de simulación meteorológica.

Referencia: Sección de clasificación de software, categoría "Aplicaciones web".

---

14. ¿Qué es la flexibilidad en términos de calidad de software?

A) La capacidad de adaptarse a otras plataformas.
B) **La facilidad para modificar el software en el mantenimiento adaptativo.**
C) La capacidad de realizar tareas en tiempo real.
D) La facilidad de reusar partes del software en desarrollos futuros.

Referencia: Factores de calidad del software, flexibilidad.

---

15. ¿Qué función tiene el software de línea de productos?

A) Controla el funcionamiento en dispositivos empotrados.
B) **Proporciona funcionalidad generalizada para una amplia variedad de usuarios.**
C) Ejecuta simulaciones avanzadas y cálculos complejos.
D) Sirve como base para otros desarrolladores de aplicaciones.

Referencia: Sección de clasificación de software, categoría "Software de línea de productos".

---

16. ¿Qué es la ingeniería de software?

A) Solo programación de sistemas**.**
B) Desarrollo de software sin documentación**.**
C) **Diseño, desarrollo y entrega de software según los requisitos del cliente.**
D) Mantenimiento de sistemas operativos

Referencia: Definición de ingeniería de software.

---

17. ¿Quién es el responsable de supervisar el proyecto de ingeniería de software en su totalidad?

A) Cliente.
B) Programador.
C) **Project Manager.**
D) End User.

Referencia: Tabla de roles y responsabilidades.

---

18. Según la definición de Bauer (1972), ¿qué busca lograr la ingeniería de software?

A) Un software caro y exclusivo**.**
B) **Un software eficiente y económico.**

C) Un sistema operativo nuevo**.**

D) Un software que requiera pocas actualizaciones.

Referencia: Definiciones de ingeniería de software.

---

19. ¿Qué función desempeña el "System Architect"?

A) Escribe el código del sistema.

B) Realiza pruebas de software.

C) **Diseña el sistema y establece cómo interactúan sus componentes.**

D) Entrena a los usuarios finales.

Referencia: Tabla de roles y responsabilidades.

---

20. ¿Qué significa el principio de "Modularidad" en ingeniería de software?

A) Dividir el software en componentes pequeños y manejables.

B) Integrar todo el software en un único módulo.

C) Evitar el uso de funciones y procedimientos.

D) Aumentar el tamaño del software.

Referencia: Principios esenciales de ingeniería de software.

---

21. ¿Cuál de las siguientes fases NO es parte del proceso de desarrollo de software?

A) Análisis de requisitos.

B) Implementación.
C) **Distribución de hardware.**
D) Evolución.

Referencia: Etapas del proceso de desarrollo de software.

---

22. ¿Qué describe el principio de "Anticipación al cambio"?

A) Mejorar la eficiencia del sistema**.**
B) Planificar para manejar posibles cambios en el software.
C) Mantener un sistema sin cambios**.**
D) Evitar actualizaciones**.**

Referencia: Principios esenciales de ingeniería de software.

---

23. ¿Cuál es la función principal de un "Tester"?

A) Escribir código para el sistema.
B) Diseñar la arquitectura del sistema.
C) **Probar el sistema de acuerdo con los requisitos del cliente.**
D) Administrar al equipo de desarrollo

Referencia: Tabla de roles y responsabilidades.

---

24. ¿Qué etapa se realiza al inicio de un proyecto de software?

A) Evolución.
B) Codificación.

C) **Definición inicial.**
D) Implementación

Referencia: Etapas del proceso de desarrollo de software.

---

25. ¿Cuál es el objetivo de la "Gestión de la configuración del software"?

A) Codificar el software.
B) **Controlar los cambios en el software.**
C) Implementar nuevas funciones sin errores.
D) Probar el software.

Referencia: Actividades sombrilla en el desarrollo de software.

---

26. ¿Qué actividad sombrilla permite que el equipo de software monitoree el progreso del proyecto?

A) Seguimiento y control del proyecto.
B) Aseguramiento de la calidad del software.
C) Gestión de la reutilización.
D) Modelado.

Referencia: Actividades sombrilla en el desarrollo de software.

---

27. ¿Qué implica el principio de "Generalidad" en ingeniería de software?

A) Crear soluciones específicas para cada problema.
B) **Buscar soluciones reutilizables.**
C) Eliminar redundancias en el código.
D) Incrementar el tamaño del código.

Referencia: Principios esenciales de ingeniería de software

---

28. ¿Cuál es el propósito del "End User" en un proyecto de software?

A) Gestionar el proyecto.
B) Realizar las pruebas de sistema.
C) **Consumir el producto final.**
D) Diseñar el sistema.

Referencia: Tabla de roles y responsabilidades.

---

29. ¿Cuál es el enfoque principal de la "Resolución de problemas" en ingeniería de software?

A) Diseñar software sin requerimientos específicos.
B) **Resolver un problema específico planteado por el cliente.**
C) Eliminar la necesidad de cambios futuros.
D) Minimizar el costo de desarrollo.

Referencia: Descripción de la resolución de problemas en ingeniería de software.

---

30. ¿Cuál de los siguientes autores define la ingeniería de software como "el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software"?

A) Bauer (1972).
B) Boehm (1976).
C) **Zelkovitz (1978).**
D) Pressman (2005)

Referencia: Definiciones de ingeniería de software.

---

31. ¿Cuál es la principal diferencia entre la ingeniería de software y otras disciplinas de ingeniería como la civil o eléctrica?

A) La ingeniería de software utiliza menos recursos.
B) **La ingeniería de software se basa en el conocimiento como "materia prima".**
C) La ingeniería de software no requiere diseño.
D) La ingeniería de software es tangible

Referencia: "la 'materia prima' es el conocimiento" (Naturaleza Intangible del Producto y Relación con Otras Disciplinas de Ingeniería).

---

32. ¿Qué característica tiene el software una vez que se desarrolla?

A) Se desgasta físicamente con el tiempo.
B) Es difícil de replicar.
C) **Puede replicarse de manera exacta y a bajo costo.**
D) Necesita procesos de manufactura adicionales.

Referencia: "se puede replicar de manera exacta y a bajo costo" (Producción y Manufactura).

33. ¿A qué está propenso el software, a pesar de no experimentar desgaste físico?

A) Al deterioro físico.
B) **A la obsolescencia tecnológica.**
C) A una vida útil infinita.
D) A una mejora continua sin fallos

Referencia: "el software es susceptible a la obsolescencia" (Producción y Manufactura).

---

34. ¿Qué teoría ayuda a entender la evolución del software?

A) Leyes de la Física de Newton.
B) Leyes de la Electrónica.
C) **Leyes de la Evolución del Software de Lehman.**
D) Leyes de la Computación de Turing

Referencia: "las 'Leyes de la Evolución del Software' de Lehman" (Enfoque Científico en la Ingeniería de Software).

---

35. ¿Cómo puede "deteriorarse" el software a diferencia de productos físicos?

A) A través del desgaste por uso.
B) **Por la falta de actualizaciones y fallos acumulados.**
C) Por su resistencia limitada.
D) Por la pérdida de recursos tangibles.

Referencia: "puede 'deteriorarse' de manera conceptual debido a la acumulación de fallos o a la falta de compatibilidad con nuevos entornos operativos" (Mantenimiento y Evolución).

36. ¿Qué área relacionada se centra en teorías y algoritmos, a diferencia de la ingeniería de software?

A) Ingeniería de sistemas.
B) Ingeniería eléctrica.
C) **Ciencias de la computación.**
D) Matemáticas aplicadas

Referencia: "la ciencia de la computación, que se enfoca en teorías y algoritmos" (Interacción con Otras Disciplinas de la Computación y Ciencias).

---

37. ¿Cuál de las siguientes organizaciones es reconocida por promover estándares de calidad en la ingeniería de software?

A) WHO.
B) **IEEE.**
C) NASA.
D) UNICEF

Referencia: "organizaciones acreditadas, como ACM e IEEE" (La Ingeniería de Software como Profesión).

---

38. ¿Cuál de los siguientes es un desafío distintivo en el ciclo de vida del software en comparación con otras ingenierías?

A) Su duración indefinida.
B) La dificultad para replicarlo.
C) **El rápido ciclo de vida por cambios tecnológicos.**
D) La estabilidad y consistencia en el tiempo

Referencia: "el corto ciclo de vida de sus productos" (Obsolescencia y Rápido Ciclo de Vida).

---

39. ¿Qué parte de otras ingenierías ha sido adaptada para la ingeniería de software?

A) Manufactura en línea.

B) Uso de materiales tangibles.

C) **Gestión de proyectos y modelado estructurado.**

D) Construcción de puentes

Referencia: "la gestión de proyectos en ingeniería civil y el modelado estructurado se han adaptado" (Adaptación de Conocimientos y Metodologías de Otras Ingenierías).

---

40. ¿Cuál es un aspecto en el que la ingeniería de software y la ingeniería de sistemas se diferencian?

A) **La ingeniería de sistemas abarca tanto hardware como software.**

B) La ingeniería de sistemas es solo teoría.

C) La ingeniería de software no trabaja con sistemas complejos.

D) La ingeniería de software solo se enfoca en hardware

Referencia: "la ingeniería de sistemas... integra tanto hardware como software" (Interacción con Otras Disciplinas de la Computación y Ciencias).

---

41. ¿Qué aspecto ético implica proteger los datos sensibles de los usuarios en ingeniería de software?

A) Transparencia.

B) Competencia Profesional.
C) **Privacidad.**
D) Sostenibilidad

Referencia: Video del tutorial

---

42. ¿Cuál es uno de los principios clave en los métodos ágiles?

A) Desarrollo en fases finales.
B) **Entrega incremental de funcionalidades.**
C) Análisis detallado previo al desarrollo.
D) Revisión anual del producto

Referencia: Video del tutorial

---

43. ¿Qué tipo de requerimiento describe la "velocidad de respuesta" en un sistema?

A) Funcional
B) Estético
C) **No funcional**
D) Operativo

Referencia: Video del tutorial

---

44. ¿Cuál de los siguientes patrones arquitectónicos organiza el sistema en capas que se pueden modificar de manera independiente?

A) Cliente-Servidor.
B) Microservicios.
C) Modelo-Vista-Controlador (MVC).
D) **Arquitectura en capas**

Referencia: Video del tutorial

---

45. ¿Cuál de los siguientes atributos evalúa la capacidad del software para funcionar sin errores?

A) Usabilidad.
B) **Fiabilidad.**
C) Funcionalidad.
D) Eficiencia.

Referencia: Video del tutorial

---

46. ¿Qué tipo de prueba evalúa la usabilidad del sistema desde la perspectiva del usuario?

A) Pruebas de integración.
B) Pruebas de sistema.
C) Pruebas unitarias.
D) **Pruebas de usuario**

Referencia: Video del tutorial

47. ¿Cuál es el propósito principal de la administración del cambio en el desarrollo de software?

A) Crear nuevos requerimientos.
B) **Documentar y controlar cada modificación.**
C) Reescribir el código base.
D) Reducir el tiempo de desarrollo.

Referencia: Video del tutorial

---

48. ¿Qué herramienta se utiliza para rastrear y restaurar versiones anteriores del código?

A) Gestión de requerimientos.
B) **Control de versiones.**
C) Pruebas de calidad.
D) Administración de proyectos

Referencia: Video del tutorial

---

49. ¿Cuál es uno de los métodos ágiles que organiza el trabajo en "sprints"?

A) **Scrum.**
B) XP.
C) Modelo en cascada.
D) Lean

Referencia: Video del tutorial

---

50. ¿Qué atributo de calidad del software se enfoca en el uso óptimo de recursos?

A) Mantenibilidad.
B) Fiabilidad.
C) **Eficiencia.**
D) Funcionalidad.

Referencia: Video del tutorial

6. CONCLUSIÓN

La ingeniería de software representa una disciplina única dentro del ámbito de la ingeniería debido a su enfoque en productos intangibles, la ausencia de procesos de manufactura tradicionales y su necesidad de adaptación constante a nuevas tecnologías. Aunque se beneficia de principios y metodologías de otras ingenierías, esta área ha desarrollado sus propias prácticas para responder a desafíos específicos, integrando un enfoque científico y profesional que la distingue como una disciplina tanto técnica como conceptual.

Imagen obtenida de Internet; derechos de autor pueden aplicarse.
IMÁGEN 8. Persona trabajando de forma organizada y correcta.

7. BIBLIOGRAFÍA

1. Aparicio, A. (2012). Ingeniería de software. EN: Data teca, Universidad Nacional Abierta y a Distancia.

2. Briano, C. (2023). Compilación de apuntes sobre conceptos fundamentales de la Ingeniería de Software. (2da ed.). Cesar Ariel Briano.

3. Gómez, M., Cervantes, J., & Gonzáles, P. (2019). Fundamentos de Ingeniería de Software. Universidad Autónoma Metropolitana.

4. Gómez, S., & Moraleda, E. (2020). Aproximación a la Ingeniería del Software. (2da ed.). Universitaria Ramón Areces.

5. Pradel, J., & Raya, J. (2020). Introducción a la ingeniería del software. (4ta ed.). Fundació Universitat Oberta de Catalunya (FUOC).

6. Pressman, R. (2010). Ingeniería del software. Un enfoque práctico. (7ma ed.). University of Connecticut.

7. Roger, Y. (2013). Software Engineering: A Hands-On Approach. Atlantis.

8. Sánchez, S., Sicilia, M. Á., & Rodríguez, D. (2012). Ingeniería del software. Un enfoque desde la guía SWEBOK. (19 ed.). Alfaomega Grupo Editor.

9. Sommerville, I. (2005). Ingeniería del software. (7ma ed.). Pearson Educación.

10. Sommerville, I. (2011). Ingeniería de software. (99 ed., L. M. Cruz Castillo, Ed.). Pearson Educación de México.

11. Tsui, F., Karam, O., & Bernal, B. (2014). Essentials of Software Engineering.Jones & Bartlett Learning.

12. Von Mayrhauser, A. (1990). Software engineering: Methods and management. Academic Press.

13. Wang, Y., & King, G. (2000). Software engineering processes: Principles and applications. CRC Press LLC.